Adding Parallelism to HPC Applications using Reveal

DOE Centers of Excellence Performance Portability Meeting

Heidi Poxon
Technical Lead
Programming Environment
Cray Inc.

April 2016

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

When to Move to a Hybrid Model

- **When code is network bound**
  - Increased MPI collective and point-to-point wait times
- **When MPI starts leveling off**
  - Too much memory used, even if on-node shared communication is available
  - As the number of MPI ranks increases, more off-node communication can result, creating a network injection issue
- **When contention of shared resources increases**

Approach to Adding Parallelism

1. **Identify key high-level loops**
   - Determine where to add additional levels of parallelism
2. **Perform parallel analysis and scoping**
   - Split loop work among threads
3. **Add OpenMP layer of parallelism**
   - Insert OpenMP directives
4. **Analyze performance for further optimization, specifically vectorization of innermost loops**
   - We want a performance-portable application at the end

The Problem – How Do I Parallelize This Loop?

- How do I know this is a good loop to parallelize?
- What prevents me from parallelizing this loop?
- Can I get help building a directive?

```
subroutine sweepz
…
do j = 1, js
 do i = 1, isz
   radius = zxc(i+mypez*isz)
   theta  = zyc(j+mypey*js)
   do m = 1, npez
    do k = 1, ks
     n = k + ks*(m-1) + 6
     r(n) = recv3(1,j,k,i,m)
     p(n) = recv3(2,j,k,i,m)
     u(n) = recv3(5,j,k,i,m)
     v(n) = recv3(3,j,k,i,m)
     w(n) = recv3(4,j,k,i,m)
     f(n) = recv3(6,j,k,i,m)
    enddo
   enddo
   …
   call ppmlr
   do k = 1, kmax
     n = k + 6
     xa  (n) = zza(k)
     dx  (n) = zdz(k)
     xa0(n) = zza(k)
     dx0(n) = zdz(k)
     e   (n) = p(n)/(r(n)*gamm)+0.5 &
        *(u(n)**2+v(n)**2+w(n)**2)
   enddo
   call ppmlr
…
 enddo
enddo
```

```
subroutine ppmlr

call boundary
call flatten
call paraset(nmin-4, nmax+5, para, dx, xa)

call parabola(nmin-4,nmax+4,para,p,dp,p6,pl,flat)
call parabola(nmin-4,nmax+4, para,r,dr,r6,rl,flat)
call parabola(nmin-4,nmax+4,para,u,du,u6,ul,flat)

call states(pl,ul,rl,p6,u6,r6,dp,du,dr,plft,ulft,&
            rlft,prgh,urgh,rrgh)
call riemann(nmin-3,nmax+4,gam,prgh,urgh,rrgh,&
             plft,ulft,rlft pmid umid)
call evolve(umid, pmid)  ← contains more calls

call remap ← contains more calls

call volume(nmin,nmax,ngeom,radius,xa,dx,dvol)

call remap ← contains more calls

return
end
```

COMPUTE | STORE | ANALYZE

Hybridization Step 1: Loop Work Estimates

Gather loop statistics using CCE and the Cray performance tools to determine which loops have the most work

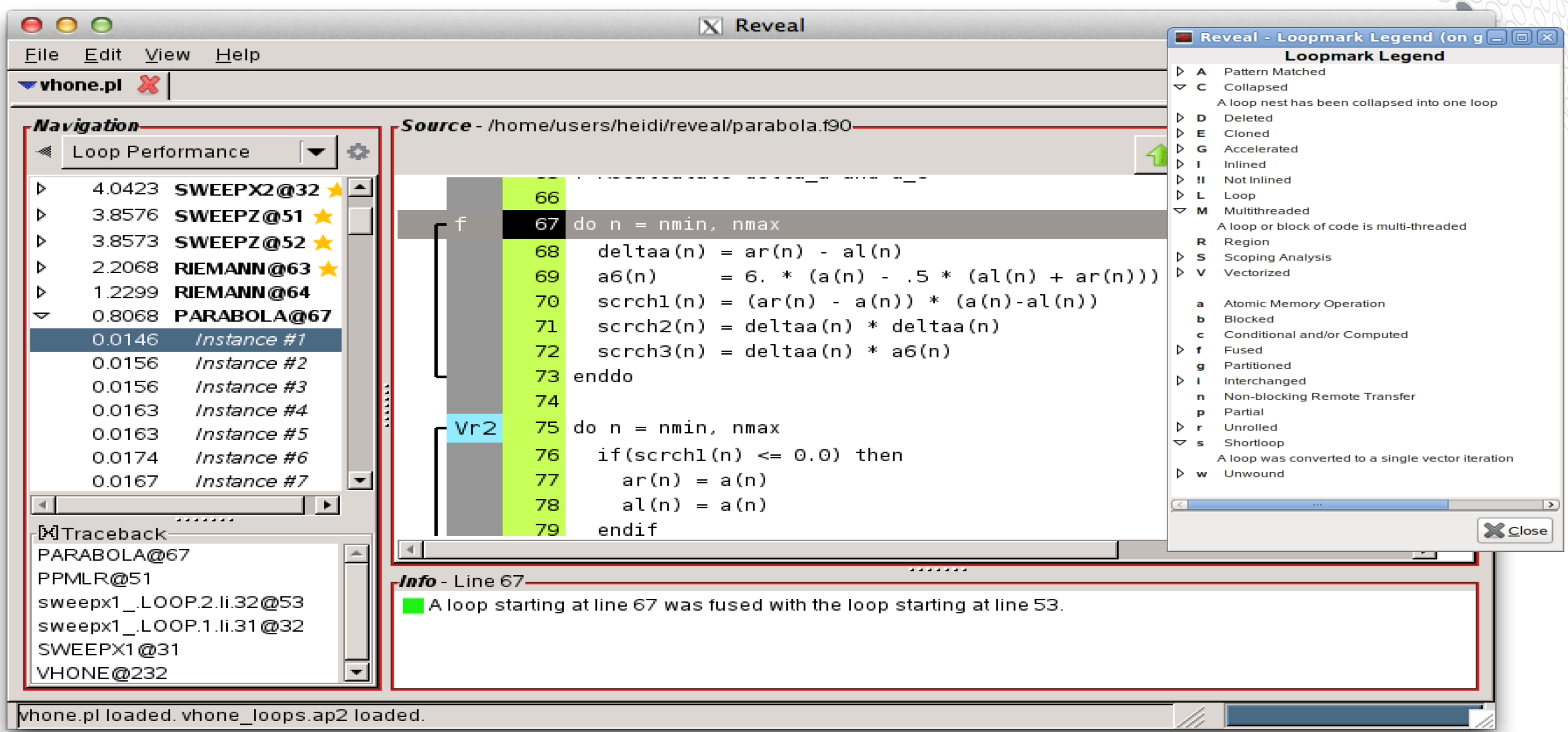
- **Helps identify high-level serial loops to parallelize**
  - Based on runtime analysis, approximates how much work exists within a loop
- **Provides the following statistics**
  - Min, max and average trip counts
  - Inclusive time spent in loops
  - Number of times a loop was executed

Example Loop Work Estimates

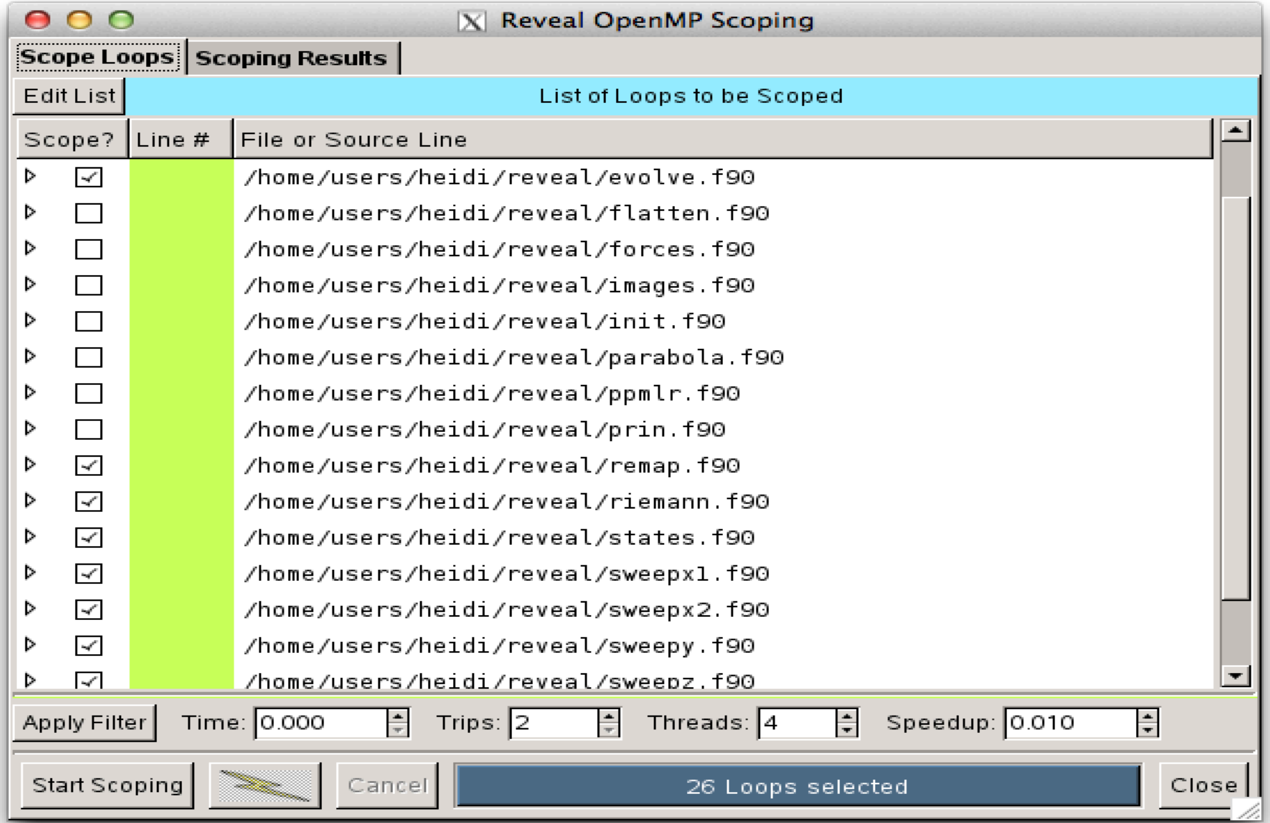
Table 2: Loop Stats by Function (from -hprofile_generate)

Loop Incl Time Total	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
8.995914	100	25	0	25	sweepy_.LOOP.1.li.33
8.995604	2500	25	0	25	sweepy_.LOOP.2.li.34
8.894750	50	25	0	25	sweepz_.LOOP.05.li.49
8.894637	1250	25	0	25	sweepz_.LOOP.06.li.50
4.420629	50	25	0	25	sweepx2_.LOOP.1.li.29
4.420536	1250	25	0	25	sweepx2_.LOOP.2.li.30
4.387534	50	25	0	25	sweepx1_.LOOP.1.li.29
4.387457	1250	25	0	25	sweepx1_.LOOP.2.li.30
2.523214	187500	107	0	107	riemann_.LOOP.2.li.63
1.541299	20062500	12	0	12	riemann_.LOOP.3.li.64
0.863656	1687500	104	0	108	parabola_.LOOP.6.li.67

View Source and Optimization Information

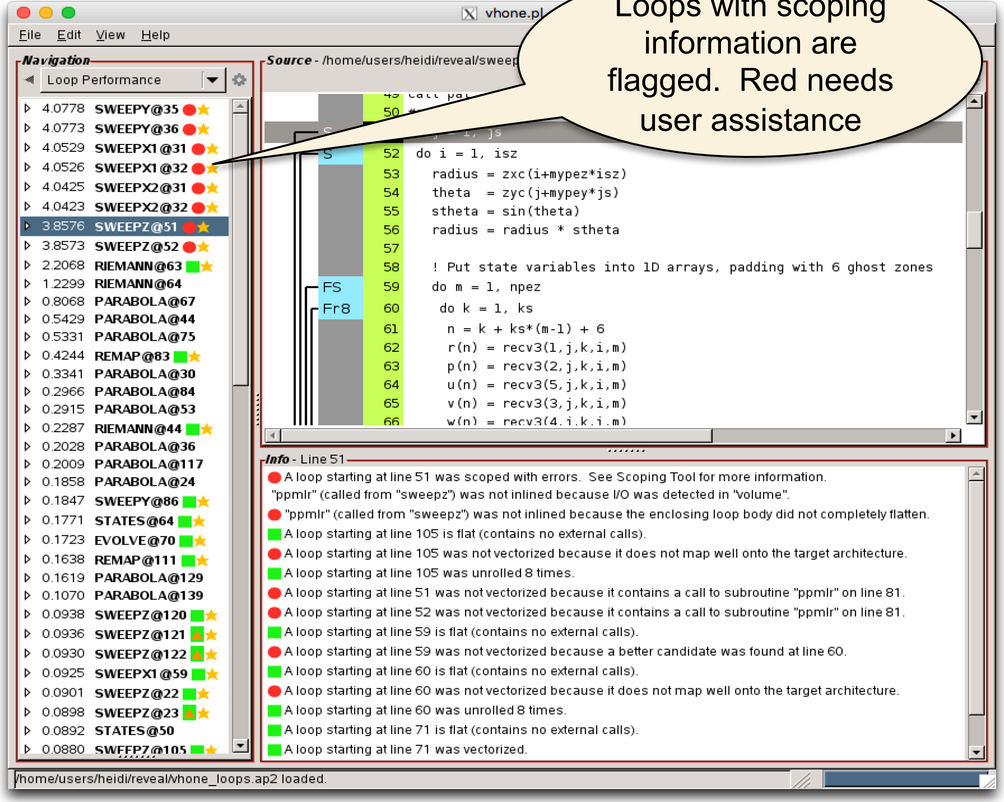


Hybridization Step 2: Scope Selected Loop(s)

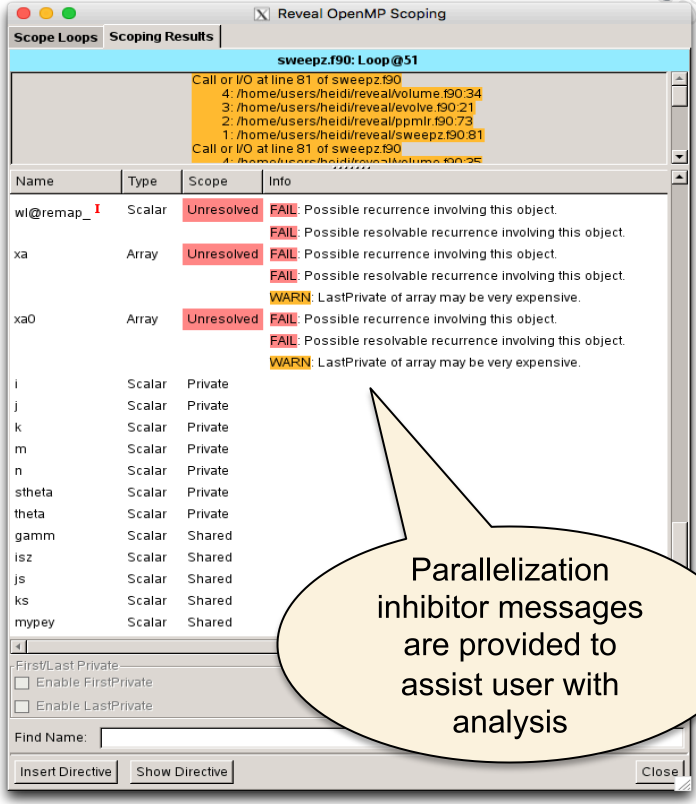


COMPUTE | STORE | ANALYZE

Review Scoping Results

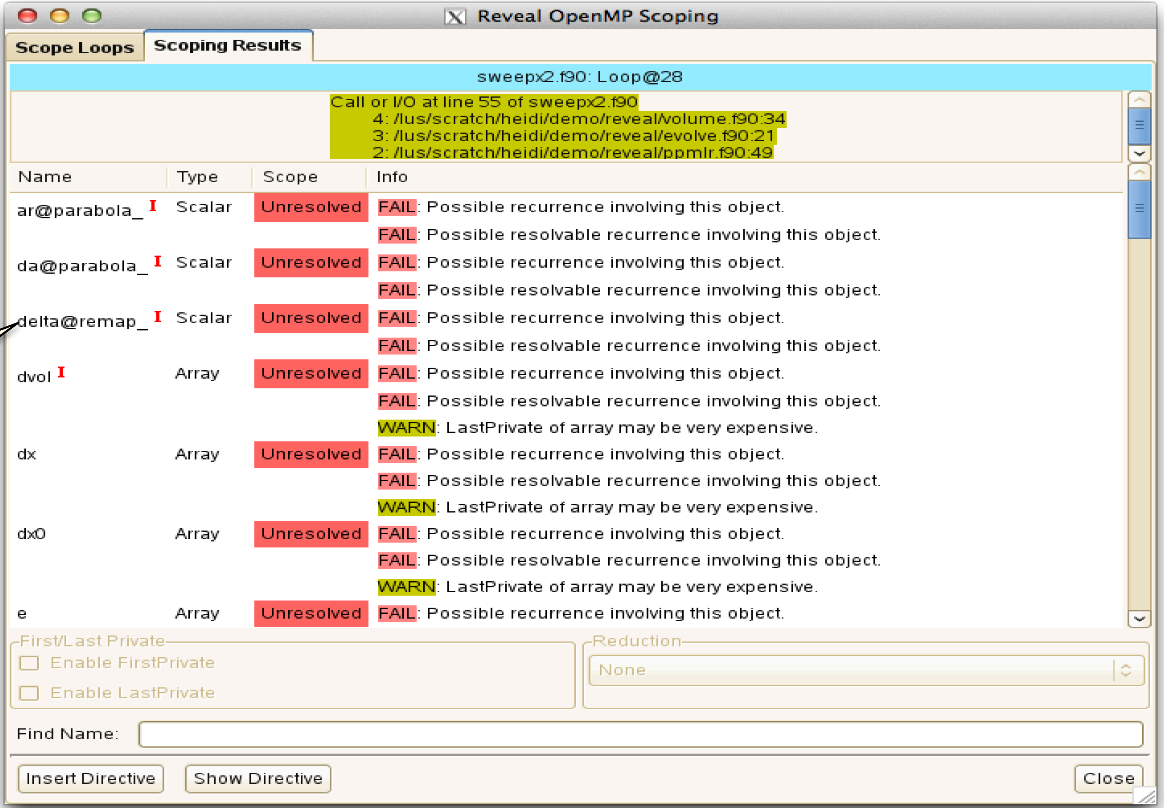


Loops with scoping information are flagged. Red needs user assistance



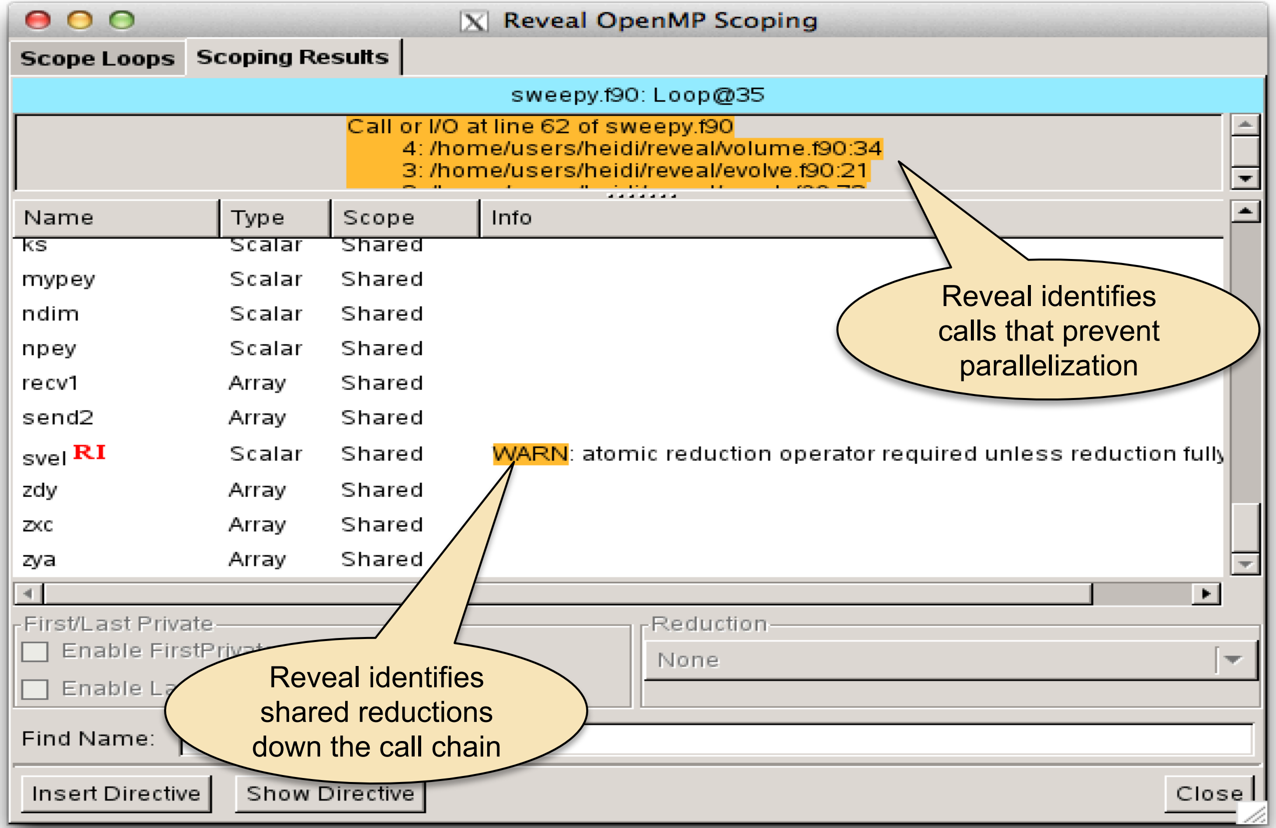
Parallelization inhibitor messages are provided to assist user with analysis

Review Scoping Results (2)



‘I’ identifies variables that reside in functions within the loop

Review Scoping Results (3)



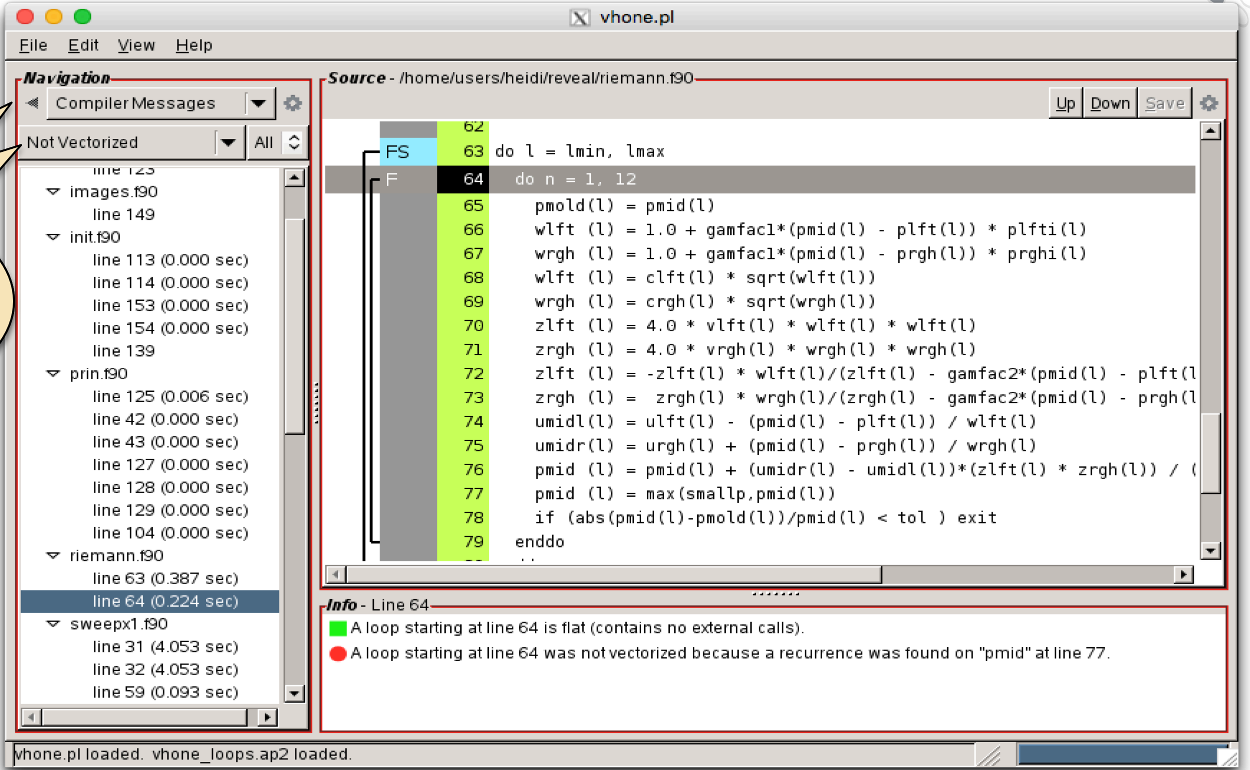
Hybridization Step 3: Generate OpenMP Directives

```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)                                          &
!$OMP&   unresolved (dvol,dx,dx0,e,f,flat,p,para,q,r,radius,svel,u,v,w,   &
!$OMP&           xa,xa0)                                                  &
!$OMP&   private (i,j,k,m,n,$$_n,delp2,delp1,shock,temp2,old_flat,        &
!$OMP&           onemfl,hdt,sinxf0,gamfac1,gamfac2,dtheta,deltx,fractn,   &
!$OMP&           ekin)                                                    &
!$OMP&   shared  (gamm,isy,js,ks,mypey,ndim,ngeomy,nlefty,npey,nrighty, &
!$OMP&           recv1,send2,zdy,zxc,zya)
do k = 1, ks
 do i = 1, isy
   radius = zxc(i+mypey*isy)

   ! Put state variables into 1D arrays, padding with 6 ghost zones
   do m = 1, npey
    do j = 1, js
     n = j + js*(m-1) + 6
     r(n) = recv1(1,k,j,i,m)
     p(n) = recv1(2,k,j,i,m)
     u(n) = recv1(4,k,j,i,m)
     v(n) = recv1(5,k,j,i,m)
     w(n) = recv1(3,k,j,i,m)
     f(n) = recv1(6,k,j,i,m)
    enddo
   enddo

   do j = 1, jmax
     n = j + 6
```

Reveal generates OpenMP directive with illegal clause marking variables that need addressing

Hybridization Step 4: Performance Analysis

Choose “Compiler Messages” view to access message filtering, then select desired type of message



Summary

- **Reveal can be used to simplify the task of adding OpenMP to MPI programs**
- **The result is performance portable code**
  - Programs can be built with any compiler that supports OpenMP
- **Can be used as a stepping stone for codes targeted for nodes with higher core counts and as the first step in adding directives to applications to target GPUs**